APPENDIX "A"

```
/* ***************************************************************/
/* ***** Copyright 1999 Veridicom, Inc. All rights reserved. ******/
/* ***************************************************************/



/* Fast template for matching on a smartcard */

#define FAST_TMPL_NUM_ENTRIES 100
#define FAST_TMPL_NUM_NEIGHS 15

typedef struct fastTemplateEntry_s {
   short minuInfo; /* contains quantized x,y, angle info */
   char neighInfo[FAST_TMPL_NUM_NEIGHS][3];
      /* contains quantized distance, relAngle, relAngDiff info */
} FAST_TEMPLATE_ENTRY;

typedef struct fastTemplateHdr_s {
   char version;
   char numMinu;
} FAST_TEMPLATE_HDR;

typedef struct fastTemplate_s {
   FAST_TEMPLATE_HDR header;
   FAST_TEMPLATE_ENTRY constellation[FAST_TMPL_NUM_ENTRIES];
} FAST_TEMPLATE;

#define XDIFF_THRESH 4
#define YDIFF_THRESH 3
#define ANGDIFF_THRESH 7
#define DIST_DIFF 1
#define RELANGLE_DIFF 6
#define RELANGLEDIFF_DIFF 6

char fastMatchConstellationCenter(
   FAST_TEMPLATE_ENTRY *constellationA,
   FAST_TEMPLATE_ENTRY *constellationB );
```

```
char fastMatchConstellationNeighs(
   FAST_TEMPLATE_ENTRY *constellationA,
   FAST_TEMPLATE_ENTRY *constellationB,
   char numNeighsB );

long fastMatch(
   FAST_TEMPLATE *templateA,
   FAST_TEMPLATE *templateB )
{
   /* templateA is on smartcard. templateB is from reader. */

   char i, j;
   char score;
   char tmp;
   char numNeighsB;
   score = 0;

   for (i = 0; i < templateB->header.numMinu; i++)
   {
      /* need to find out how many actual neighbors there are */
      numNeighsB = FAST_TMPL_NUM_NEIGHS;
      for (j = 0; j < FAST_TMPL_NUM_NEIGHS; j++)
      {
         /* calculate once ..,not every time in inner neighbor loop */

         if (templateB->constellation[i].neighInfo[j][0] == 0 &&
            templateB->constellation[i].neighInfo[j][1] == 0 &&
            templateB->constellation[i].neighInfo[j][2] == 0 &&
            j > 6)
         {
            numNeighsB = j;
            break;
         }
      }

      /* ok ... now compare against other template */
      for (j = 0; j < templateA->header.numMinu; j++)
      {
         tmp = fastMatchConstellationCenter(
            &(templateA->constellation[j]),
            &(templateB->constellation[i]) );

         if (tmp == 0) continue;
```

```
            else if (tmp < 0) break;

            score += fastMatchConstellationNeighs(
               &(templateA->constellation[j]),
               &(templateB->constellation[i]),
               numNeighsB);
         }

         if (score > 25) break;
      }

      return(score);
   }

   char fastMatchConstellationCenter(
      FAST_TEMPLATE_ENTRY *constellationA,
      FAST_TEMPLATE_ENTRY *constellationB)
   {
      char diff, diff2;

      diff = (
         ((char) ((constellationA->minuInfo&0Xf000)>>12)) -
         ((char) ((constellationB->minuInfo&0xf000)>>12)) );

      if ((diff) > XDIFF_THRESH)
         return( (char) -1); /* due to sort, can stop comparing
            templateB against any more templateA's */

      if ((diff) < -XDIFF_THRESH ) return( (char) 0);
         /* still need to Compare templateB against any more templateA's */

      diff2 =
         ((char) ((constellationA->minuInfo&0x0f00)>>8)) -
         ((char) ((constellationB->minuInfo&0X0f00)>>8));

      if ((diff2) > YDIFF_THRESH && diff == XDIFF_THRESH ) return( (char) -2);
         /* due to sort, can stop comparing templateB against any more templateA's */

      if ((diff2) < -YDIFF_THRESH) return( (char) 0); /* still need to compare templateB
against any more templateA's */
```

```
    diff =
        ((char) ((constellationA->minuInfo&0x00ff))) -
        ((char) ((constellationB->minuInfo&0x00ff)));

    if (diff < 0) diff = -diff;

    if ((diff) > ANGDIFF_THRESH && (diff) < 115 ) return( (char) 0);

    return( (char) 1 );
}

char fastMatchConstellationNeighs(
    FAST_TEMPLATE_ENTRY *constellationA,
    FAST_TEMPLATE_ENTRY *constellationB,
    char numNeighsB)
{
    char diff;
    char i, j;
    char jstart;
    char *pNeighInfoA, *pNeighInfoB;
    char score;

    /* initial limits to be refined as we go */
    jstart = 0;
    score = 0;
    for (i = 0, pNeighInfoA = constellationA->neighInfo[0];
        i < FAST_TMPL_NUM_NEIGHS;
        i++, pNeighInfoA += 3) /* move to next neighbor */
    {
        if (pNeighInfoA[0] == 0 &&
            pNeighInfoA[1] == 0 &&
            pNeighInfoA[2] == 0 &&
            i > 6 ) break;

        for (j = jstart, pNeighInfoB = constellationB->neighInfo[jstart];
            j < numNeighsB; j++, pNeighInfoB += 3 /* move to next neighbor */ )
        {
            diff = pNeighInfoA[0] - pNeighInfoB[0];
            if (diff < 0)
            {
                diff = -diff;
                if (diff > DIST_DIFF) /* if go too far, stop. No hope. */
                {
```

```
                if (j > 0) jstart = j-1;
                break;
            }

        }
        else
        {
            if (diff > DIST_DIFF) /* if go too far, just try next one */
                continue;
        }

        diff = (pNeighInfoA[1] - pNeighInfoB[1]);

        if (diff < 0) diff = -diff;
        if (diff > RELANGLE_DIFF && diff < 120)
        {
            continue;
        }

        diff = (pNeighInfoA[2] - pNeighInfoB[2]);
        if (diff < 0) diff = -diff;
        if (diff > RELANGLEDIFF_DIFF && diff < 120)
        {
            continue;
        }

        score++;
    }
}

if (score > 6) score = 1;
else score = 0;

return(score);
}
```